4 - Tipi di dato primitivi

Programmazione e analisi di dati
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼milazzo
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica
A.A. 2014/2015

Tipi (1)

Negli esempi precedenti abbiamo definito variabili di tipo `int` e `double`

```
int base=5;
final double tasso=0.05;
```

La specifica del tipo di una variabile consente al compilatore di effettuare controlli sull'uso di tali variabili

Ad esempio, il comando

```
base=3.5;
```

viene segnalato come un errore dal compilatore, il quale risponde con il seguente messaggio

```
Prova.java:7: error: possible loss of precision
base=3.5;
     ^
  required: int
  found:    double
1 error
```

Tipi (2)

Il controllo dei tipi effettuato dal compilatore serve a prevenire errori di programmazione

Se dichiariamo una variabile di un certo tipo e poi le assegnamo valori di un altro tipo, forse le stiamo assegnando il valore sbagliato!

Inoltre, sapere che una variabile ha un certo tipo offre al programmatore delle garanzie. Ad esempio:

```
int x,y,z;

.....

System.out.print("Risultato: ");
System.out.println(x*2+(y/4-7)*z*z);
```

Qualunque siano i valori assegnati a `x`, `y` e `z` il risultato sarà un numero intero.

Tipi primitivi (1)

I tipi primitivi del linguaggio Java sono i seguenti

Nome tipo	Tipo di valore	Memoria usata	Valori consentiti
`byte`	Intero	1 byte	da -128 a 127
`short`	Intero	2 byte	da -32.768 a 32.767
`int`	Intero	4 byte	da -2.147.483.648 a 2.147.483.647
`long`	Intero	8 byte	da -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
`float`	Numero in virgola mobile (precisione singola)	4 byte	da $-3.40282347 \times 10^{38}$ a $3.40282347 \times 10^{38}$
`double`	Numero in virgola mobile (precisione doppia)	8 byte	da $-1.7*10^{308}$ a $1.7*10^{308}$
`char`	Carattere singolo	2 byte	Tutti i caratteri della codifica Unicode (65.535 caratteri)
`boolean`	Valore di verità	1 bit	`true` o `false`

NOTA: I valori di tipo intero sono memorizzati in complemento a due...

Tipi primitivi (2)

Tra i tipi numerici, `int` e `double` sono quelli usati più comunemente

I tipi `byte`, `short` e `float` di solito si usano solo quando si hanno problemi di occupazione di memoria o di prestazioni di calcolo

Il tipo `long` di solito si usa solo quando bisogna rappresentare numeri interi molto grandi (miliardi)

Il tipo `char` serve per rappresentare singoli caratteri

Il tipo `boolean` serve per effettuare scelte che condizionano l'esecuzione del programma

Tipi primitivi (3)

Per gli scopi di questo corso NON UTILIZZEREMO i tipi `byte`, `short` e `float`

OCCASIONALMENTE potremmo usare il tipo `long`

I tipi di dato primitivi che UTILIZZEREMO principalmente saranno invece:

`int`, `double`, `char` e `boolean`

I tipi `char` e `boolean` li vedremo più avanti!

Compatibilità di assegnamento

Il compilatore deve controllare le operazioni di assegnamento

Un'operazione di assegnamento ha la forma

variabile = *espressione;*

Ad esempio:

```
raggio = 12;
area1 = 12*12*3.14;
area2 = raggio*raggio*3.14
```

In ogni operazione di assegnamento il compilatore deve controllare

che il tipo del valore che risulta dall'espressione
sia compatibile con il tipo della variabile

Tipo di una espressione (1)

Qual è il tipo del valore che risulta da un espressione?

- Più brevemente, qual è il tipo di una espressione?

Consideriamo i tipi numerici `int`, `long` e `double`

Consideriamo le espressioni più semplici: i letterali

- Il letterale 35 fa parte dei valori consentiti per il tipo `int`, ma anche `long` e `double`
- Il letterale 5000000000l fa parte dei valori consentiti per il tipo `long`, ma anche `double` (nota[1])
- Il letterale 3.5 fa parte solo dei valori consentiti per il tipo `double`

[1] nei letterali interi maggiori di 2147483647 e minori di -2147483648 bisogna apporre la "l" finale (come long)

Tipo di una espressione (2)

L'esempio dei letterali mostra che:

- un'espressione può essere compatibile con diversi tipi
- esiste una relazione di ordinamento $\rightarrow$ tra i tipi `int`, `long` e `double` tale che:

  $$\texttt{int} \rightarrow \texttt{long} \rightarrow \texttt{double}$$

  che esprime la possibilità di convertire il valore di un tipo nel valore di un'altro tipo in maniera implicita (automatica)
- La freccia $\rightarrow$ significa "esiste una conversione implicita da .. a .."
- Può esistere una conversione implicita da un tipo a un altro quando tutti i valori accettabili del primo sono accettabili anche per il secondo
- Ovviamente, vale `int` $\rightarrow$ `double` (la relazione è transitiva)

Tipo di una espressione (3)

Ma allora.... qual è il tipo di un'espressione?

Risposta (un po' empirica):

Il tipo di un'espressione artimetica
è il minimo tipo (rispetto la relazione $\rightarrow$)
che accetta tutti gli elementi che la costituiscono

Esempi:

- `7+8+(23*324)/22-((21*53)/11)` è di tipo `int`
- `5+(3.2*(4-3))` è di tipo `double`
- Assumendo

```
int i;
long l;
double d;
```

  abbiamo
  - `12+(l-i)` è di tipo `long`
  - `l+i+d` è di tipo `double`
  - `i+(i-1)` è di tipo `int`

Compatibilità di assegnamento

E in un assegnamento

$$variabile = espressione;$$

quando il tipo dell'espressione è compatibile con il tipo della variabile?

Risposta:

In un assegnamento il tipo dell'espressione
deve essere convertibile implicitamente nel tipo della variabile

Esempi:

```
int somma=12+21; //corretto!
long totale=somma; //corretto!
double supertot=totale; //corretto!
int megatot=supertot+totale; //sbagliato!
```

Cosa risponde il compilatore?

```
Prova.java:7: error: possible loss of precision
int megatot=supertot+totale; //sbagliato!
                     ^
  required: int
  found:    double
1 error
```

Esempi

Quali di questi assegnamenti sono corretti?

```
int i1,i2;
long l1,l2;
double d1,d2;

i1 = 3.5*10;
i2 = 340/10+(13*2);
l1 = 188/10+i;
d1 = i1+l1;
l2 = d1+123*(7-1);
d2 = 0.001*(d1+l2)*1000;
```

Osservazione:

- tutte queste espressioni danno un risultato intero,
- ma per stabilire se un assegnamento è corretto conta il tipo dell'espressione, non il suo risultato.

Conversioni di tipo (type cast) (1)

Le regole di compatibilità degli assegnamenti consentono di convertire automaticamente un `int` in un `double`.

```
int i = 100;
double d = i;
```

E se avessi bisogno di convertire un `double` in un `int`?

```
double prezzo = 25.99;
int numeroeuro = prezzo; // assegnamento ILLECITO!
```

Si può forzare una conversione (type cast) in questo modo

```
double prezzo = 25.99;
int numeroeuro = (int)prezzo; // assegnamento LECITO!
```

La trasformazione da `double` a `int` è ottenuta troncando il numero

- si eliminano le cifre decimali: 25.99 diventa 25

Conversioni di tipo (type cast) (2)

Il type cast forza il cambiamento di tipo dell'espressione che la segue

Esempi:

```
int I=1000;
long L=5000;
double D=7.0;

I=(int)(d*0.5)          // risultato: I=3
I=(int)L                // risultato: I=5000
I=(int)50000000001      // ris: impredicibile (troppo grande per int)
L=(long)(d*0.5)         // risultato: L=3
D=(double)(5/2)         // risultato: D=2.0
D=(double)5/(double)2   // risultato: D=2.5
D=(double)5/2           // risultato: D=2.5 -- come D=((double)5)/2
```